Available online at www.sciencedirect.com

ScienceDirect

Computational Geometry 40 (2008) 231–251

Computational Geometry
Theory and Applications

www.elsevier.com/locate/comgeo

Iterated snap rounding with bounded drift ☆

Eli Packer

School of Computer Science, State University of New York at Stony Brook, New York, USA

Received 27 March 2007; accepted 24 September 2007

Available online 10 October 2007

Communicated by M. de Berg

Abstract

Snap Rounding and its variant, Iterated Snap Rounding, are methods for converting arbitrary-precision arrangements of segments into a fixed-precision representation (we call them SR and ISR for short). Both methods approximate each original segment by a polygonal chain, and both may lead, for certain inputs, to rounded arrangements with undesirable properties: in SR the distance between a vertex and a non-incident edge of the rounded arrangement can be extremely small, inducing potential degeneracies. In ISR, a vertex and a non-incident edge are well separated, but the approximating chain may drift far away from the original segment it approximates. We propose a new variant, Iterated Snap Rounding with Bounded Drift, which overcomes these two shortcomings of the earlier methods. The new solution augments ISR with simple and efficient procedures that guarantee the quality of the geometric approximation of the original segments, while still maintaining the property that a vertex and a non-incident edge in the rounded arrangement are well separated. We investigate the properties of the new method and compare it with the earlier variants. We have implemented the new scheme on top of CGAL, the Computational Geometry Algorithms Library, and report on experimental results.

Keywords: Robustness; Geometric rounding; Line segments; Arrangements

1. Introduction

Implementation of geometric algorithms is generally difficult because one must deal with both precision problems and degenerate input. While these issues are usually ignored when describing geometric algorithms in theory, overlooking them may result in program crashes and wrong results. A variety of techniques have been developed to cope with these difficulties [21].

Finite-Precision-Approximation is an approach whose idea is to convert the representation of the input into finite-precision. The goal is to make the representation more robust for algorithms that follow (by that we mean that those algorithms will be less likely to produce wrong computations due to the lack of exact-precision datatypes). Certain conditions should hold to ensure robustness and efficiency. For example, important requirements from such methods

☆ A preliminary version of this work appeared in the Proceedings of the Twenty-Second Annual Symposium on Computational Geometry, 2006, pp. 367–376.

Work on this paper has been partially supported by the National Science Foundation (CCR-0098172, CCF-0431030).

E-mail address: epacker@cs.sunysb.edu.

0925-7721/$ – see front matter
doi:10.1016/j.comgeo.2007.09.002

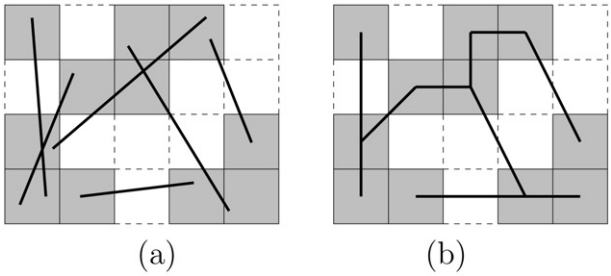

Fig. 1. An arrangement of segments before (a) and after (b) Snap Rounding (hot pixels are shaded).

could be small deviation of the input and topology preservation. For a survey on Finite-Precision-Approximation see, e.g., [20].

Snap Rounding (SR, for short), is a well known Finite-Precision-Approximation method, mainly devoted to arrangement of line segments in the plane. It converts the input segments into polygonal chains in the following way. The plane is tiled with a grid of unit squares, *pixels*, each centered at a point with integer coordinates.[1] A pixel is *hot* if it contains vertices of the arrangement. Then the output of each segment is the polygonal chain through the centers of the hot pixels met by it in the order along the segment. See Fig. 1 for an illustration.

In Snap Rounding both the deviation of the input is very small and certain (limited) topological properties are preserved.[2] However, in a previous work [14] we showed that a vertex of the output may be extremely close to a non-incident edge,[3] inducing potential degeneracies. (In general, determining which side of the edge the vertex lies on may require double precision and some more computations than the ISR, which requires single precision.) We proposed an augmented procedure, Iterated Snap Rounding (ISR, for short), aimed to eliminate this undesirable property. The output of ISR is equivalent to the final output of a finite series of applications of SR. It rounds the arrangement differently from SR, such that any vertex is at least half-the-width-of-a-pixel away from any non-incident edge. Fig. 2 depicts an example in which SR introduces a short distance between a vertex and a non-incident edge. This distance is increased by ISR as illustrated in this example. ISR, however, may round segments far from their origin. Fig. 2(c) illustrates long drift from the input. In [19] we gave a pathological example where the approximating segment is $\Theta(n^2)$ units away from the original segment (the example in Fig. 2 has a similar structure). Fig. 3 illustrates the differences between SR and ISR too (borrowed from [14]).

1.1. Our contribution

We propose a new algorithm, Iterated Snap Rounding with Bounded Drift (ISRBD, for short), which rounds the segments such that both the distance between a vertex and a non-incident edge is at least half-the-width-of-a-pixel, and the deviation is bounded by a user-specified parameter. ISRBD is a modification of ISR. What we do is take the ISR algorithm and plug in two new procedures.

In the first (*GenerateNewHotPixels*), new hot pixels are introduced. These hot pixels are used to bound the rounding magnitude. This procedure is the heart of this paper. We show that in theory, and practice, the space required for the output is not significantly larger than the one required by ISR. ISRBD, therefore, eliminates the undesirable feature of ISR (namely the possible large deviation), which in turn maintains a half-unit distance between vertices and non-incident edges.

The second procedure (*RemoveDegree2Vertices*) comes to improve the output quality by removing some of the degree-2 vertices as we discuss next. Berg et al. [5] proved that degree-2 vertices which do not correspond to endpoints can be safely removed without violating the properties SR possess. This removal can be viewed as merging the two

[1] It is important to note that pixels are closed only at two non-parallel sides (say left and bottom), not including two of the corners (upper-left and bottom-right in this case).

[2] An important property that the SR preserves is that no segment ever crosses completely over a vertex of the arrangement.

[3] The distance between a vertex and a non-incident edge can be made as small as $1/\sqrt{(2^b-1)^2+1} \approx 2^{-b}$, where b is the number of bits in the representation. The SR/ISR tiling contains $2^b \times 2^b$ unit pixels.

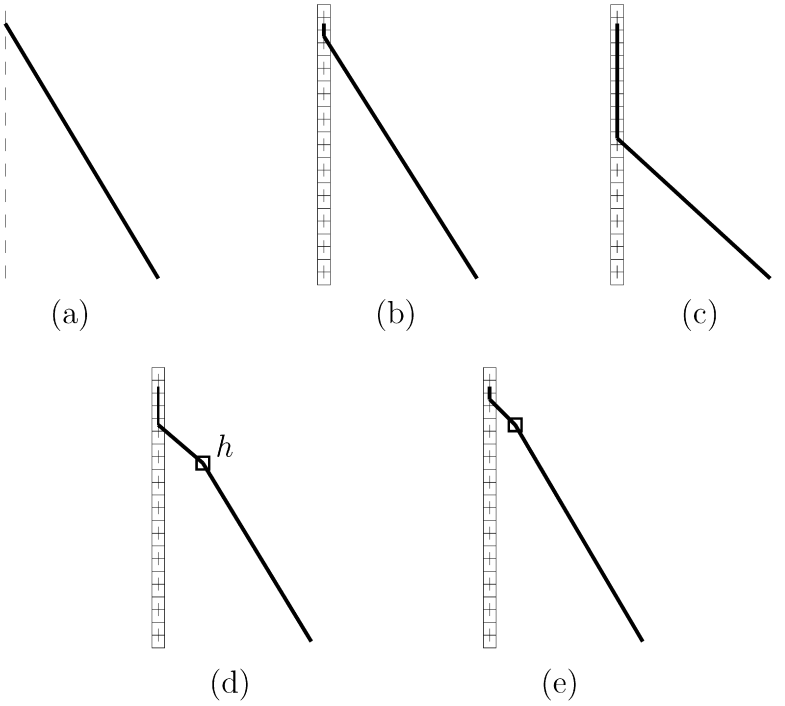

Fig. 2. Results of SR, ISR and ISRBD of an example with large drift. All squares are hot pixels. The edges of new hot pixels and the 'interesting' segment are marked by bold lines. (a) Input segments (b), SR output, (c) ISR output, (d)–(e) ISRBD output for two different values of δ.

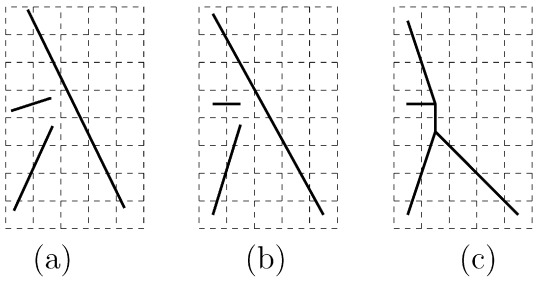

Fig. 3. An arrangement of segments before (a), after Snap Rounding (b) and after Iterated Snap Rounding (c).

adjacent links[4] of the corresponding vertex by connecting the neighbor vertices with a replacing link. In this work we use the same ideas, but need not remove special kinds of degree-2 vertices that are necessary for maintaining the properties of ISRBD (see Section 3.2). We further show that applying this procedure may significantly improve the output.

Both procedures are fairly simple to understand and implement and use well known geometric data structures (the simplicity depends on having these data structure available).

We augment a list of desirable properties presented in [5] and show that as opposed to SR and ISR, ISRBD satisfies all. Thus, we believe that ISRBD may be a good option to choose when a snap-rounded–like arrangement of segments is required.

[4] In the rest of this paper, by *link* we refer to a single line segment that may be a part of a polysegment of the output or that is generated in an intermediate step of the algorithm.

1.2. Related work

Greene and Yao [11] proposed a rounding scheme that preceded Snap Rounding. Several algorithms were proposed to implement Snap Rounding [3,9,10,12,16,17]. They were mainly devoted to improve asymptotic time complexity of the process, to present on-line algorithms and to extend Snap Rounding to 3D. Recently, de Berg et al. [5] introduced an improved output-sensitive algorithm for Snap Rounding and an output optimization which is used with changes in our work. We [14] introduced the Iterated Snap Rounding which is a variant of Snap Rounding that keeps vertices and non-incident edges well separated. Eigenwillig et al. [8] extended Snap Rounding to Bezier curves. Other perturbation schemes aimed for robustness are presented in [7,13,15,18].

The rest of the paper is organized as follows. In Section 2 we present the main ideas of this work. In Section 3 we describe our algorithm. Implementation details and complexity analysis of our algorithm are given in Section 4. In Section 5 we list desirable properties that a snap-rounded–like arrangement should satisfy and fit them to ISR and ISRBD. In Section 6 we present experiments performed with our implementation. We conclude and present ideas for future research in Section 7.

2. Preliminaries and key ideas

In order to make our description and analysis more clear, we normalize the input such that the pixel edge size is *unit length*. We use the following notations throughout the paper. $S = \{s_1, s_2, \ldots, s_n\}$ is the set of input segments. Let $A(S)$ be the arrangement of S with output $A^*(S)$. Let S^* be the list of output chains. Let $H = \{h_1, h_2, \ldots, h_m\}$ be the set of hot pixels. Since H is dynamic in ISRBD in a sense that hot pixels can be inserted and removed on the fly, H refers to the set that exists at the time of reference, unless otherwise stated. Let $s \in S$ be a segment in the input and $h \in H$ be a hot pixel. From now on, whenever we refer to s or h without explicitly defining them, we mean that s is any segment of S and h is any hot pixel of H. Let $h(p)$ be the hot pixel containing a point p. For any pixel h, let h_c be its center with coordinates h_x and h_y. We denote the output of s by $\lambda(s)$. Let $d(s)$ be the Hausdorff distance between s and $\lambda(s)$. Our goal, therefore, is to bound $d(s)$. For any set of one or more geometric objects, denoted by X, let $B(X)$ be the axis-aligned bounding box of X. We denote the triangle with vertices x, y and z by $\Delta(x, y, z)$.

For each hot pixel h and segment s such that h properly intersects $B(s)$ (their interior intersect), we define a right triangle $\Delta(s, h)$ (the definition applies when s has a negative slope and it is to the right of h; other cases are defined analogously). Let v_1 be the upper-right corner of h, ℓ be the infinite line containing s and $\vec{v}$ be the vector $[\frac{1}{2}, \frac{1}{2}]$. Let ℓ' be the line parallel to ℓ, shifted by $\vec{v}$ units from ℓ. Let v_2 and v_3 be the intersections between ℓ' and the horizontal and vertical rays from v_1 towards ℓ'. Then $\Delta(s, h) \equiv \Delta(v_1, v_2, v_3)$. See Fig. 4 for an illustration.

We define the following predicates:

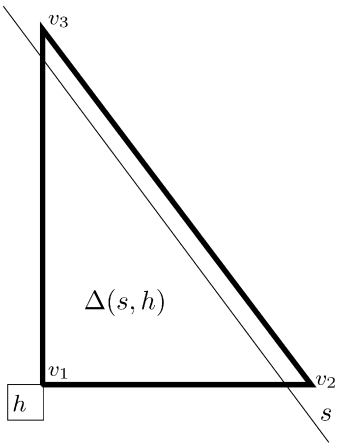

Fig. 4. $\Delta(s, h)$ is the triangle whose edges are marked by bold lines.

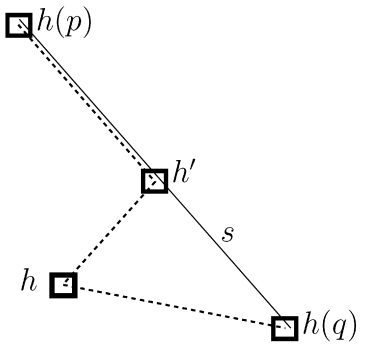

Fig. 5. A segment s (the solid line) and a possible chain through the centers of hot pixels $h(p)$, h', h and $h(q)$ (the dashed polygonal chain). Such a chain through these hot pixels is never weakly-monotone.

- For any segment s and a hot pixel h, $\Xi(s,h)$ is true if and only if $\Delta(s,h)$ contains no center of hot pixels in its interior.
- For any two hot pixels h_1 and h_2, $\Phi(h_1,h_2)$ is true if and only if their centers have either the same x-coordinate or the same y-coordinate.
- For any segment s and two hot pixels h and h', $\zeta(h',h,s)$ is true if and only if the center of h' is contained inside $\Delta(s,h)$.

We continue by introducing a few properties that hold for ISR (Lemmata 2.1–2.3).

Lemma 2.1. *$\lambda(s)$ is weakly-monotone.*[5]

Proof. Suppose that $\lambda(s)$ is not weakly-monotone. Recall that ISR is equal to a series of SR applications. It follows that during at least one of the iterations of SR, a link ℓ is snapped by a hot pixel h_2, creating two links that are not weakly-monotone. Let h_1 and h_3 be the hot pixels containing the endpoints of ℓ. We get a contradiction since it follows that a straight segment ℓ intersects both h_1, h_2 and h_3, while the chain through their centers is not weakly-monotone. □

Lemma 2.2. *$\lambda(s)$ lies within $B(h(p)_c, h(q)_c)$ where p and q are the endpoints of s.*

Proof. Suppose that $\lambda(s)$ is not entirely within $B(h(p)_c, h(q)_c)$. Then there is a hot pixel, h, such that the polygonal chain through $h(p)_c$,h_c and $h(q)_c$ is not weakly-monotone. It follows that $\lambda(s)$ is not weakly-monotone in contradiction to Lemma 2.1. □

For the next lemma we restrict ourselves to hot pixels that properly intersect $B(s)$. Others are irrelevant for this discussion since $\lambda(s)$ does not contain their centers as Lemma 2.2 indicates.

Lemma 2.3. *For any segment $s \in S$ and hot pixel $h \in H$, h_c cannot belong to $\lambda(s)$ if $\Xi(s,h)$ is false.*

Proof. Without loss of generality, assume that h lies to the left of s, and that s has a negative slope. Suppose the claim is false and $\Delta(s,h)$ contains a center of a hot pixel h' and $\lambda(s)$ contains h_c. Let p and q be the endpoints of s. According the definition of $\Delta(s,h)$, h_c and h'_c cannot be located within different sides of s. Since the rounding magnitude of each iteration is at most $\frac{\sqrt{2}}{2}$ units, the output link cannot snap to h_c without snapping to h'_c as well (or in other words, it cannot bypass h'_c). It follows that h_c, h'_c, $h(p)_c$ and $h(q)_c$ are all vertices of $\lambda(s)$. However, these four vertices cannot form a weakly-monotone subsequence in $\lambda(s)$, in contradiction to Lemma 2.1. See Fig. 5 for an illustration. □

[5] A curve is said to be *weakly-monotone* if its slope is either never negative or never positive with respect to the coordinate system.

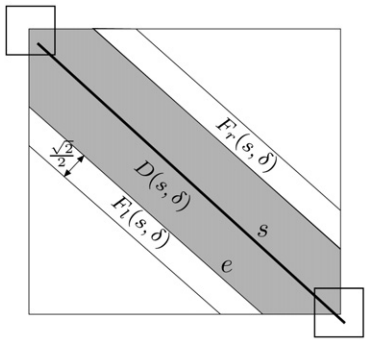

Fig. 6. The domain $D(s,\delta)$ (shaded) and forbidden locus $F(s,\delta) = F_l(s,\delta) \cup F_r(s,\delta)$. The input s is the thick segment crossing $D(s,\delta)$. The two small squares are the hot pixels containing the endpoints of s.

The main idea of our work is to bound the deviation of the output of ISR. Let δ be the maximum deviation allowed, given as a parameter. We require that $\delta > \frac{3\sqrt{2}}{2}$ for a reason described in Section 3.1. For each segment $s \in S$ with endpoints p and q, δ defines a domain $D(s,\delta)$ that must contain $\lambda(s)$ (see Fig. 6). Since $\lambda(s)$ lies inside $B(h(p)_c, h(q)_c)$ (Lemma 2.2), $D(s,\delta)$ is the intersection of $B(h(p)_c, h(q)_c)$ with the Minkowski sum of s and a circle with radius δ centered at the origin. We ignore the case where s is either horizontal or vertical, since in this case $d(s) \leqslant \frac{\sqrt{2}}{2}$, maintaining a deviation that is clearly smaller than δ (see Section 4.4 for more details).

Our goal, therefore, is to bound $\lambda(s)$ to $D(s,\delta)$. Let $F_l(s,\delta)$ and $F_r(s,\delta)$ be the two trapezoids, lying to the left and right of $D(s,\delta)$, respectively. $F_l(s,\delta)$ is defined as follows. One edge of $F_l(s,\delta)$ is the non-rectilinear left edge of $D(s,\delta)$, denoted by e (see Fig. 6). Let $\vec{v}$ be the vector $[\frac{-1}{2}, \frac{-1}{2}]$. The opposite edge of e in $F_l(s,\delta)$ is parallel to e and shifted $\vec{v}$ units from e; its endpoints lie on $B(h(p)_c, h(q)_c)$. The two edges are connected by segments of $B(h(p)_c, h(q)_c)$. $F_r(s,\delta)$ is defined similarly to the right of $D(s,\delta)$. See Fig. 6 for an illustration. Let $F(s,\delta) = F_l(s,\delta) \cup F_r(s,\delta)$ and $F(S,\delta) = \{F(s,\delta) \mid s \in S\}$. The following claims refer only to the area lying to the left of s, but are symmetric and can be applied to the right.

We mentioned that ISR is equivalent to the final output of a finite series of applications of SR. Consider some segment s for which the output is obtained after N applications of SR. Let $\phi_i(s)$ be the temporary chain of s after the ith application of SR. Since links are rounded to centers of hot pixels, the Hausdorff distance between $\phi_i(s)$ and $\phi_{i+1}(s)$ is at most $\frac{\sqrt{2}}{2}$ units for any $1 \leqslant i \leqslant N-1$. Notice that if $\lambda(s)$ is snapped during one of the applications of SR to the left (right) beyond $D(s,\delta)$, then the center of the hot pixel responsible for that snapping must lie within $F_l(s,\delta)$ ($F_r(s,\delta)$). Following the construction of $F_l(s,\delta)$ ($F_r(s,\delta)$), other hot pixels would be too far to snap any segment inside $D(s,\delta)$. Thus, in order to bound $\lambda(s)$ to $D(s,\delta)$, it is sufficient to make sure that no hot pixels whose centers are contained inside $F(s,\delta)$ ever snap s. This, in turn, is achievable if we make sure that for any hot pixel h and segment s, if h_c lies inside $F(s,\delta)$, then $\Delta(s,h)$ contains at least one center of a hot pixel due to Lemma 2.3.

The above discussion leads to the main idea ISRBD: for any segment s and hot pixel h, if $h_c \in F(s,\delta)$ and $\Xi(s,h)$ is true, we heat a pixel h' whose center lies within $\Delta(s,h)$. As a result, $\Xi(s,h)$ becomes false. Clearly, this process must be performed before the rounding stage. By modifying one of the main proofs that deals with the topology of the output [12], heating pixels as we propose may change the output, but not the topology as defined there (see Property 5.2). This process may cascade as the new hot pixel may lie within a forbidden locus of another segment.

The following corollary establishes the condition that is sufficient for bounding the deviation of $\lambda(s)$ from s:

Corollary 2.4. *If $\Xi(s,h)$ is false for each segment $s \in S$ and a hot pixel $h \in H$ whose center lies within $F(s,\delta)$, then the deviation of ISRBD is bounded by δ.*

We distinguish between pixels that are heated because they contain vertices of $A(S)$ and pixels that are heated to bound the deviation as explained above. We refer to the pixels in the first group as the *original* hot pixels, as they are defined in ISR, and to the ones in the second group as the *new* hot pixels, as they are defined in this work for the first time.

In the remainder of this work we will encounter situations in which a hot pixel h' blocks another hot pixel h from rounding a segment s. This is because h'_c is contained within $\Delta(s,h)$. In such cases, no action has to be performed even if $h_c \in F(s,\delta)$. The predicate ζ (see above) was defined to formalize this situation.

3. Algorithm

The main idea of our algorithm follows the discussion in the previous section. The idea is to heat pixels iteratively until the condition in Corollary 2.4 is met for all hot pixels and segments, namely for each segment s and a hot pixel h whose center lies within $F(s,\delta)$, $\Xi(s,h)$ is false. As a result, the output drift obtained in the rounding stage will be bounded as required.

The algorithm for computing ISR has three stages [14]. In the first, the hot pixels are detected and stored. In the second, a range search data structure is built for answering queries that report the hot pixels that a given segment intersects. The third is the rounding stage. ISRBD applies all three stages of ISR, as well as plugging in two new procedures. In the first, *GenerateNewHotPixels*, pixels to bound the drift are heated. The procedure is executed after the first stage of ISR (detecting the set of original hot pixels). It is executed after this stage because the list of original hot pixels has to be available. It is executed before the other two stages of ISR because clearly they require a complete list of hot pixels (original and new). In the second procedure, *RemoveDegree2Vertices*, some of the degree-2 vertices of the output are removed to improve the output quality. It is executed after the third stage of ISR (the rounding stage), because the output chains of all input segments have to be available. We describe both procedures in this section.

The following is a high level pseudo-code of ISRBD.

ITERATED SNAP ROUNDING WITH BOUNDED DRIFT
Input: a set S of n segments and maximum deviation δ
Output: a set S^* of n polygonal chains
1. Compute the set H of hot pixels
2. Call *GenerateNewHotPixels*
3. Construct a segment intersection search structure D on H /* H here includes the new hot pixels heated in *GenerateNewHotPixels* */
4. Perform the iterative rounding stage
5. Call *RemoveDegree2Vertices*

3.1. Heating new hot pixels

In this section we discuss the procedure *GenerateNewHotPixels*. For any segment s and hot pixel h, let s' be any segment that intersects $\Delta(s,h)$ (such a segment must exist since s itself intersects $\Delta(s,h)$ in our definition). Let $I(s',s,h)$ be the intersection of s' and $\Delta(s,h)$. Suppose a center of a hot pixel h_c lies within $F(s,\delta)$ where $\Xi(s,h)$ is true. We need to heat a pixel whose center is contained inside $\Delta(s,h)$. Next we explain which pixel we heat. Without loss of generality, we assume that h lies to the lower-left of s, where s has a negative slope. Other cases are similar. We first check if there are segments that penetrate the pixel h' with center coordinates (h_x+1, h_y+1); h cannot be in their forbidden loci since $\delta > \frac{3\sqrt{2}}{2}$ (this is the reason for constraining δ to be at least this magnitude; also see Fig. 7 for an illustration). If there are such segments, we heat this pixel (see Fig. 8(a)). Otherwise, let t be the first segment to the right of h which intersects $\Delta(s,h)$. We heat the pixel that contains the middle point of $I(t,s,h)$. Fig. 8(b) illustrates this situation in which a pixel on s is heated. Fig. 8(c) illustrates the same case, but here we heat a pixel on a segment s', which is closer to h than s. Note that no two segments may intersect inside $\Delta(s,h)$; otherwise $\Xi(s,h)$ would be false. We choose this technique for heating pixels in order to have certain claims hold (see Section 4 Once a pixel is heated, it is possible that it is located inside a forbidden locus of another segment. Thus, this process is performed for each hot pixel, whether original or new.

Our algorithm proceeds as follows. For each hot pixel $h \in H$ (original or new; the new hot pixels are inserted to H during this process), we locate the set of segments S' such that h is within $F(s,\delta)$ for each $s \in S'$. For each $s \in S'$, we check if $\Delta(s,h)$ is empty of centers of hot pixels. If it is empty, we heat a pixel as explained above.

We use a trapezoidal decomposition of forbidden loci (denoted by Γ_1), for all input segments. This data structure helps us to efficiently identify in which forbidden loci each hot pixel is located. We also use a trapezoidal decomposi-

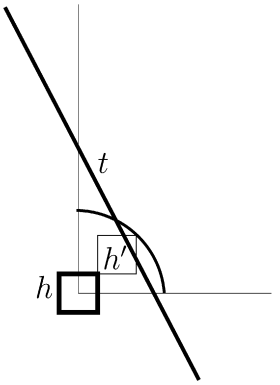

Fig. 7. h' is contained inside the quarter-circle centered at h_c with radius $\frac{3\sqrt{2}}{2} < \delta$. Since t intersects h', h cannot be in a forbidden locus of t.

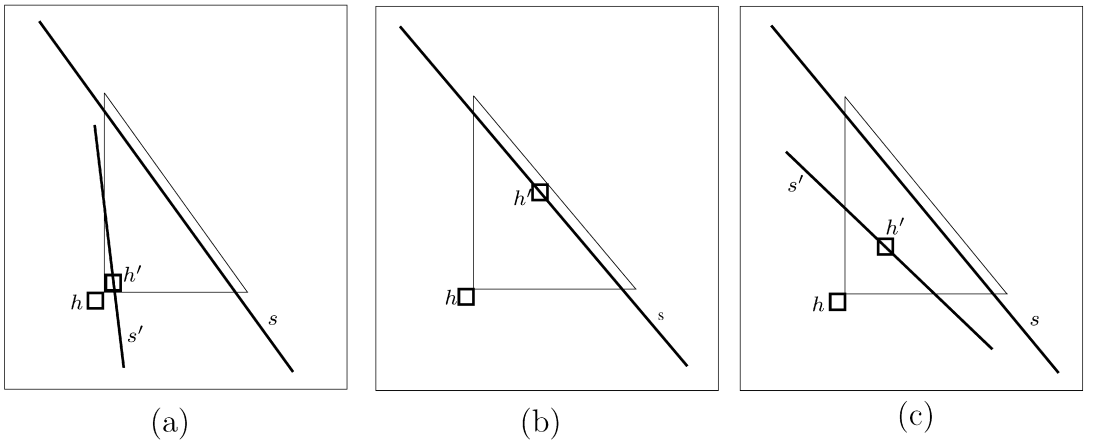

Fig. 8. Heating a new hot pixel, h' as a result of hot pixel h being within the forbidden locus of s. $\Delta(s,h)$ is depicted with thin edges in all sub-figures. (a) Heating the upper-right neighbor of h. (b) Heating on s. (c) Heating on a segment that intersects $\Delta(s,h)$.

tion of input segments (denoted by Γ_2) to be able to query segments which intersect $\Delta(s,h)$ efficiently. The purpose is to locate the segment on which we heat a pixel as described above. In order to traverse the hot pixels (original and new), we use a hot pixel queue Q, initialized to the original hot pixels. We dequeue hot pixels from Q and process them as explained above. When heated, new hot pixels are queued to be processed later. For querying the emptiness of $\Delta(s,h)$, we use a dynamic simplex range searching data structure (denoted by Ψ). It is initialized to the original hot pixels and updated whenever a pixel is heated. We use Ψ also in *RemoveDegree2Vertices*; see Section 3.2. In Section 4.2, we provide details about Γ_1, Γ_2 and Ψ.

The following theorem summarizes the quality of the geometric-approximation obtained with ISRBD.

Theorem 3.1. *The polygonal chain $c(s) \in S^*$ of any segment $s \in S$ lies inside the Minkowski sum of s with a disc of radius δ, centered at the origin.*

Proof. It immediately follows from our algorithm: whenever there is a hot pixel h in $F(s,\delta)$, we make sure that s never snaps to the center of h since we guarantee that $\Xi(s,h)$ is false after processing *GenerateNewHotPixels*. Thus, s never snaps to the centers of hot pixels which are beyond $D(s,\delta)$. It follows that only hot pixels within distance at most δ from s may be snapping points for s and the claim follows. □

The following is a pseudo-code of it GenerateNewHotPixels.

GenerateNewHotPixels
1. Build Γ_1 and Γ_2
2. Build Ψ
3. Initialize a queue, Q, with H
4. **while** Q is not empty
5. **let** $h = \text{dequeue}(Q)$
6. **foreach** segment s for which $F(s, \delta)$ contains h_c
7. **if** $\Xi(s, h)$ is true
8. Heat a pixel h' whose center is contained inside $\Delta(s, h)$ (as described in Section 3.1)
9. Queue(Q, h') and insert h'_c to Ψ
10. **end if**
11. **end foreach**
12. **end while** newline

3.2. Removing redundant degree-2 vertices

In this section we discuss the procedure *RemoveDegree2Vertices*. De Berg et al. [5] proposed to modify $A^*(S)$ by removing degree-2 vertices that do not correspond to endpoints. It is done as a post-processing step by removing them with their adjacent links and connecting their two neighboring vertices with a new link. They proved that important properties of SR still hold after this process. However, this idea will not work as is in ISRBD. Consider Fig. 2(d). The center of the new hot pixel (denoted by h) is a degree-2 vertex that would be removed if the above method is applied. However, removing h will cause the output to penetrate one of the hot pixels and not pass through its center. It is also possible that this removal will cause the output drift to be too large.

Nevertheless, applying a method in this spirit can be very beneficial in ISRBD since intuitively many of the new hot pixel centers are degree-2 vertices. Thus, we modify this method such that it will retain the properties of ISRBD. What we do is constrain the removal to only non-endpoint degree-2 vertices whose removal does not generate a link l that induces the ISRBD violations. More specifically, we pose the following two requirements: (a) l will not penetrate a hot pixel and not pass through its center. (b) The Hausdorff distance between l and its original segment will not exceed δ. We note that such this process makes the output simpler and more efficient. In Section 4.1 we analyze the output of ISRBD and show that there are examples for which this method significantly improves the quality of the output.

We denote by K the list of degree-2 vertices in the output that do not correspond to endpoints. Let $\kappa = |K|$. K can be computed easily by traversing the output arrangement. We traverse all hot pixels in K and remove hot pixels which do not violate the properties of ISRBD as listed above. Let $\beta(h)$ be the link obtained when removing a hot pixel h. It is possible that some hot pixel h which has already been processed without being removed will be eligible for removal later. There are two possible cases for that. The first is after the removal of one of its neighbors (where one of the links attached to h is changed). The second may occur if $\beta(h)$ penetrates one or more hot pixels and not passing through their centers, and then those hot pixels are removed, allowing h to be removed too. For each hot pixel, we hold a list of other hot pixels that may be redundant after its removal, as described above. We denote this list by P; each hot pixel will have its own instance. Thus, when removing a hot pixel, we take care of reprocessing the hot pixels in its instance of P.

For each $h \in K$, we check whether its removal violates the properties of ISRBD. If it does not, we remove it from H and update S^* and Ψ (Ψ was defined in Section 3.1) accordingly. The first test is done by computing the Hausdorff distance between the potential new link and the corresponding input segment and then comparing it to δ. The second test is done by checking if the potential new link does not penetrate any hot pixel without passing through its center. In order to query the hot pixels penetrated by segments (the second test), we need a dynamic range search data structure. It must be dynamic because hot pixels are removed in this routine. We use the data structure, called Ψ, which is defined and used in Section 3.1.

Remark. Since this routine can be applied in ISR as well, it would be interesting to describe how it is adjusted for the sake of ISR. The variant of ISR is very similar to the one of ISRBD, with the only difference that the check for large drift is omitted. Other than that, the algorithm and its analysis are similar.

The following is a pseudo-code of *RemoveDegree2Vertices*.

RemoveDegree2Vertices
1. Initialize every hot pixel to be *active* and a list P, one for each hot pixel, to contain the pixel neighbors
2. **foreach** $h \in K$
3. **if** h is removable (as described in Section 3.2)
4. **foreach** hot pixel h' in $h.P$
5. **if** h' is *non-active*
6. set h' to be *active*
7. move h' to the end of K /* done in order to recheck h' for redundancy */
8. **end if**
9. **end foreach**
10. remove h from K and update Ψ and the output S^* accordingly
11. **else**
12. mark h as *non-active*
13. **foreach** degree-2 hot pixel h'' which is penetrated by the potential link obtained by the removal of h, not through the center of h''
14. push h to $h''P$.
15. **end foreach**
16. **end if**
17. **end foreach**

3.3. Illustrative example

Fig. 2 illustrates the differences among SR, ISR, and ISRBD, for an input set of segments (Fig. 2(a)) where ISR incurs large drift (on one of the segments). The input is in the spirit of the large-drift example given in [19]. In this example, only one segment s will be modified in the rounding, and it is drawn in bold line. In the SR output (Fig. 2(b)), $\lambda(s)$ penetrates a hot pixel but does not go through its center, making the polygonal chain rather close to the endpoint of the other segments in that pixel—in general the polygonal chain and the endpoint of the other vertex may get extremely close to one another in the output of SR. Fig. 2(c) demonstrates the output of ISR for the same input, where the drift of the segment is large—in general the approximating chain may drift $\Theta(n^2)$ pixels away in the output of ISR. The last two figures, Figs. 2(d) and (e), show the output of ISRBD for two different values of δ: the drift is bounded as desired, and the chain and any other non-incident vertices are well separated.

4. Complexity analysis and implementation details

4.1. Output complexity

In this section we analyze the output complexity of ISRBD. We prove that for any segment $s \in S$, $\lambda(s)$ may consist of $O(n^2)$ links, resulting in an overall complexity of $O(n^3)$. This result states that although we introduce new hot pixels, the output complexity of ISRBD is equivalent to SR and ISR. The idea of the proof is that although the output may contain $O(n^3)$ new hot pixels, $\lambda(s)$ consists of $O(n^2)$ centers of such hot pixels, for each $s \in S$. It follows that the overall complexity is $O(n^3)$. Since the lower bound is $\Omega(n^3)$ due to an example presented in [14], the maximum output complexity is $\Theta(n^3)$ (this example has the same output with SR, ISR and ISRBD—the proof for that claim would be immediate). We note that if we only consider the complexity of the rounded arrangement without counting the multiplicities of overlapping segments, the output complexity of SR and ISR becomes $\Theta(n^2)$. Under this condition, we have not established tight bound for ISRBD yet. We show later that the lower bound is equal to SR and ISR, $\Omega(n^2)$, and it is at most $O(n^3)$. We also show that unless we apply *RemoveDegree2Vertices*, there exists an example that demonstrates an output complexity of $\Omega(n^3)$ even when overlapping segments are not counted. This example demonstrates the usefulness of *RemoveDegree2Vertices*.

For each hot pixel h, we denote by $f(h)$ the pixel that was the cause for heating h by being located within a forbidden locus of some segment (we also say that h is the *child* of $f(h)$ or $f(h)$ is the *father* of h). Since there

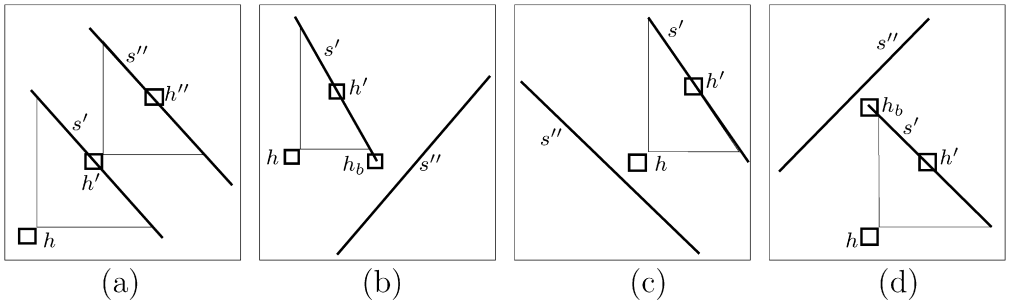

Fig. 9. Different cases of pixel heating: (a) h'' is to the upper-right of h'. (b)–(d) h'' cannot be placed on any s'' that lies to the lower-right, lower-left and upper-left of h', respectively.

is no reason to heat a pixel which is already hot, the relation induced by f defines a directed forest in which the root of each tree is an original hot pixel, and all of the pixels heated due to this pixel are below the root in this tree. Each edge in this data structure corresponds to a hot pixel and its child. For each original hot pixel h, let $T(h)$ be its corresponding tree and $C(h)$ be the set of all the hot pixels in $T(h)$. We divide $C(h)$ into four sets corresponding to the four quadrants around h (we prove later that for any hot pixel $h' \in C(h)$, $\Phi(h, h')$ is false, namely the centers of h and h' do not lie on the same horizontal or vertical lines): we number them starting from the upper-right and go clockwise. Let $C_1(h) \subseteq C(h)$ be the list of hot pixels to the upper-right of h and $\{C_i(h) \mid i = 1 \ldots 4\}$ be the lists in clockwise order in space. In our analysis, we concentrate on the upper-right quadrant. The situation in other quadrants will be equivalent because of symmetry. We build our proofs by starting with an original hot pixel and explore its tree.

Consider an original hot pixel, h, and a segment s that intersects h. Let h' be a child of h and assume that h' is located to the upper-right of h. Let s' be the segment on which h' was heated. It follows that s' has a negative slope (since h' is to the upper-right of h). Let h'' be a child of h' (assume that such a pixel exists; even if there is no such hot pixel, all the proof that follow which does not depend on h'' will hold) and s'' be the segment on which h'' was heated. We divide the plane into four quadrants around h' and explore the possibility and the results of placing h'' in a different quadrant (see Fig. 9). We first rule out two quadrants which cannot contain h''.

Lemma 4.1. *No new hot pixel needs to be heated on the upper-left and lower-right quadrants.*

Proof. If the slope of s'' is negative, s'' cannot be the source for h''. The reason is that in this case $\Delta(s'', h')$ must be in the upper-right or lower-left quadrants. Thus, we assume that s'' has a positive slope. Next we check all the possibilities for placing s''. Clearly, s'' cannot intersect triangle $\Delta(s', h)$, otherwise $\Xi(s', h)$ would be false and h' would not be heated. Consider Figs. 9(b) and (d): s'' must intersect the line containing s', because otherwise h' is not in its axis-aligned bounding box and forbidden locus (note that s'' cannot intersect h'). Thus, there must be a pixel h_b on s' such that $\zeta(h_b, h', s'')$ holds (h_b would correspond to either an endpoint of s' or to an intersection of s' with either s'' or another segment). Thus, h_b will block h' from rounding s''. □

We continue with ruling out another quadrant.

Lemma 4.2. *No new hot pixel needs to be heated on the lower-left quadrant.*

Proof. Analogously to Lemma 4.1, we can assume that s'' has a negative slope. It also must lie to the left of h'. Following the method we use to heat pixels (Section 3.1), there are no segments to the left of h' which intersect $\Delta(s, h)$ (otherwise the new pixel, possibly not h', would correspond to another segment $\tilde{s} \neq s$). We are left with the case shown in Fig. 9(c): there is a segment s'' which lies to the lower-left of h', its axis-aligned bounding box contains h'_c and whose forbidden locus contains h'. Then $\zeta(h, h', s'')$ holds and h will block s'' from snapping to h'. Thus, no pixel has to be heated in this case.

The next lemma provides another insight to the location of h'. We prove that the center of h and h' cannot lie in the same horizontal or vertical line.

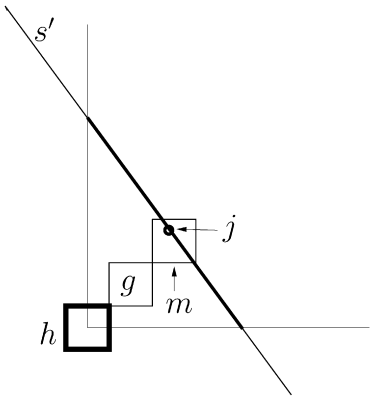

Fig. 10. j, the center of $s'' = I(s', s', h)$ (the thick segment; s'' does not intersect pixel g which is not hot), is contained inside a pixel m such that $\Phi(m, h)$ is never true. Note that following Lemma 2.2, s'' intersects the lines containing the upper and right edges of h.

Lemma 4.3. *$\Phi(h, h')$ is false.*

Proof. If we heat the upper-right neighbor of h (denote it by $\tilde{h}$), it is clear that the claim is valid. Otherwise, clearly no segment intersects $\tilde{h}$. Let $b = I(s', s', h)$ (the portion of s' on $\Delta(s', h)$). Since b does not intersect $\tilde{h}$, it extends to more than two x and y units (its projected length on both axes is greater than two units). It must also intersect both legs of $\Delta(s', h)$. Thus, the middle point of b cannot be contained inside a hot pixel h' in a way that $\Phi(h, h')$ is true (in other words, h and h' will not lie in the same horizontal or vertical line). See Fig. 10 for an illustration. □

Following the last three lemmata, we get that the only possible location for h'' is to the upper-right of h'. In this case we heat the appropriate pixel. This case is possible as illustrated in Fig. 9(a) in which h' lies within $F(s'', \delta)$. Clearly, this process of heating pixels may cascade further to the upper right in the same way. Next we prove that the constructed structure is a chain.

Lemma 4.4. *The pixels of $C_1(h)$ are arranged in a chain under h in $T(h)$.*

Proof. We prove that at most one pixel may be the child of h in the upper-right quadrant. If the upper-right neighbor of h is heated, it will clearly block any other segments in the upper-right quadrant from snapping to h, thus there is no need for another child of h. Otherwise, suppose that two distinct pixels are heated on segments s_1 and s_2 while processing h and s' (thus they are the children of h in $T(h)$). Let $I_1 = I(s_1, s_1, h)$ and $I_2 = I(s_2, s_2, h)$. I_1 and I_2 cannot intersect—otherwise their intersection would heat a pixel h' such that $\zeta(h', h, s_1)$ holds. Then, without loss of generality, let I_1 be closer to h than I_2. Following our algorithm, no pixel would be heated on I_2 because of h, regardless of the order in which we process segments. This hot pixel (possibly on s_1) will block both s_1 and s_2 from snapping to h. Thus, no pixel is heated on s_2 in contradiction to our assumption.

In the same way, we can prove that any descendant of h to the upper-right has at most one child. Due to symmetry, this behavior is similar in other three quadrants. Thus, all the pixels of $C_1(h)$ are contained in the above chain under h. □

Let $T_{ur}(h)$ be this sequence of hot pixels, starting from h towards the upper right direction.

We continue with two lemmata that constrain the output of each segment.

Lemma 4.5. *Let $s \in S$ be a segment that intersects pixel h. Then $\lambda(s)$ cannot pass through any child h' of h.*

Proof. If the slope of s is not positive, it cannot intersect h' (the weakly-monotone condition would be violated since h' is to the upper-right of h). Thus, assume that s has a positive slope. Since s intersects h, it does not penetrate $\Delta(s', h)$ (otherwise $\Xi(s', h)$ would be false). Then it must be the case that if $\lambda(s)$ penetrates $\Delta(s', h)$, it will have to be non-weakly-monotone by changing its direction towards another hot pixel h'' on s'. h'' corresponds either to an

endpoint of s' or to an intersection on s' (with s or another segment). We get that the only way for s to snap to h' is to surround $\Delta(s', h)$. However, note that for arguments similar to Lemma 4.3, $\Phi(h', h'')$ is false. Thus, $\lambda(s)$ would not be weakly-monotone if it surrounds $\Delta(s', h)$. It follows that $\lambda(s)$ will not snap to h'. □

Lemma 4.6. *For any segment $s \in S$ and original hot pixel $h \in H$, $\lambda(s)$ can pass through at most one hot pixel in $T_{ur}(h)$.*

Proof. Lemma 4.5 can be easily generalized such that any segment s which passes through one of the pixels in $T_{ur}(h)$, cannot pass through any other pixels in $T_{ur}(h)$. Since the pixels in $T_{ur}(h)$ are monotonously increasing in the y-direction, any segment with negative slope (whose output must be monotonously-decreasing in y) cannot pass through more than one of them. The claim follows. □

We are now ready to describe the structure of $T(h)$.

Lemma 4.7. *For each original hot pixel h, $T(h)$ satisfies the following properties*:

(a) *The root has at most four children.*
(b) *Each child of the root is a root of a subchain.*
(c) $|T(h)| = O(n)$.

Proof. (a) and (b) are immediate consequences of Lemma 4.4, when applied for all directions. (c) Consider $T_{ur}(h)$. We heat pixels that intersect input segments. From Lemma 4.6, each segment $s \in S$ can intersect at most one pixel of $T_{ur}(h)$. Thus s can be the source of heating at most one pixel in $T_{ur}(h)$. Then the size of $C_1(h)$ is $O(n)$. Since the situation in other quadrants is similar and from (a) and (b), $|T(h)| = O(n)$. □

We next bound the number of hot pixels created in our algorithm.

Lemma 4.8. *The overall number of hot pixels generated in GenerateNewHotPixels is* $O(n^3)$.

Proof. It immediately follows from Lemma 4.7(c), since there are $O(n^2)$ original hot pixels. □

We continue with a theorem that summarizes the output complexity.

Theorem 4.9. *The maximum output complexity of ISRBD is* $\Theta(n^3)$.

Proof. In [14] it is shown that the output may need $\Omega(n^3)$ space. The lower bound example need not heat any new hot pixels when using ISRBD (it is clear from the structure of the example; we omit the exact proof). Thus, this example provides a lower bound for ISRBD as well. Since for each segment $s \in S$, $\lambda(s)$ intersects $O(1)$ new hot pixels from $C(h)$ for each original hot pixel h (Lemma 4.6), $\lambda(s)$ may intersect $O(n^2)$ new hot pixels. It can also intersect $O(n^2)$ original hot pixels. Thus, each output chain consists of $O(n^2)$ links. Then the overall output complexity of ISRBD is $O(n^3)$. The claim follows. □

From Theorem 4.9, the output complexity is the same as SR and ISR. (Note again that if we do not count overlapping segments more than once, then the output complexity of SR and ISR decreases to $\Theta(n^2)$ while we do not know yet what the bound of ISRBD is.)

From Lemma 4.8, the overall number of pixels created in *GenerateNewHotPixels* is $O(n^3)$. Thus, the overall number of hot pixels is $O(n^3)$ as well. It is $\Omega(n^2)$ [14]. We hope to close this gap in future research. We experimented with many examples in an attempt to produce many new hot pixels, but the number of new hot pixels was always far less than the number of original hot pixels.

We next show that performing *RemoveDegree2Vertices* can be very useful.

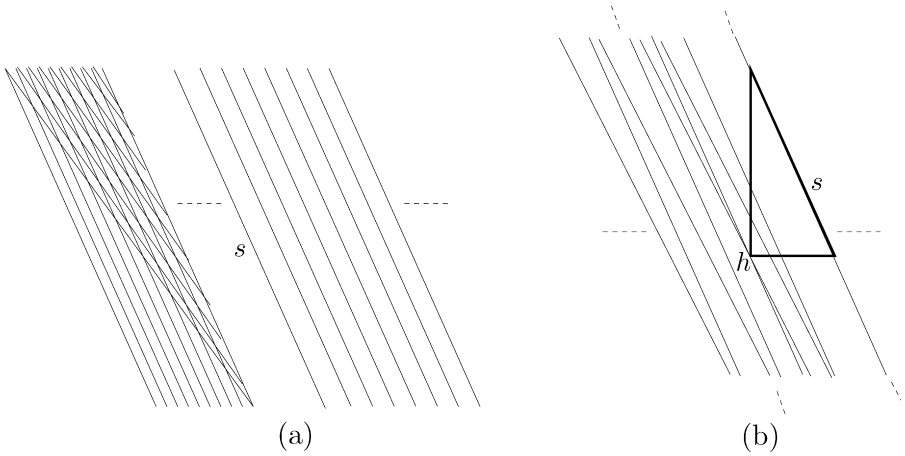

Fig. 11. An example of an arrangement of segments which is similar to another arrangement in which $\Theta(n^3)$ pixels are heated in *GenerateNewHotPixels*. The dashed lines represent gaps that contain the same pattern on the right. Subfigure (b) is a zoom-in view of (a).

Lemma 4.10. *If RemoveDegree2Vertices is not called, the number of pixels created in GenerateNewHotPixels is* $\Omega(n^3)$ *and the maximum overall output complexity is* $\Omega(n^3)$ *even if overlapping segments are not counted more than once.*

Proof. Consider Fig. 11(a): its left part consists of a grid of $n/2$ segments which includes two groups with $n/4$ segments each. The segments in each group are parallel and very close and each segment from the first group intersects each segment from the second one. Let s be the first segment to the right of the grid. Now imagine that we perturb the segments of this structure such that each hot pixel h containing that contains an intersection is far enough from the other intersections in a way that $\Delta(s,h)$ satisfies $\Xi(s,h)$. It would be impossible to illustrate this situation clearly here, instead a zoom-in view with deceptive proportions is illustrated in Fig. 11(b). In this subfigure, there is only one visible intersection, denoted by h. $\Delta(s,h)$ is drawn in bold and contains no intersections. The segments in both groups will also be very close such that all the hot pixels will be inside $F(s,\delta)$ and sorting them with respect to x-values will give a monotonically decreasing list. The second structure is composed of $n/2$ parallel segments, starting from s and continuing to the right. The distance between each two consecutive segments is such that all of the centers of all hot pixels on one segment will be within the forbidden locus of the segment to its right. It follows that $\Theta(n^2)$ pixels are heated along s and each causes a series of $\Theta(n)$ pixels to be heated on the segments to the right of s. Thus, $\Theta(n^3)$ new pixels are created. An immediate result is that the output chain of each segment in the right structure consists of $\Theta(n^2)$ links. Since no output chain of such share any link with other output chains, the overall output complexity is $\Omega(n^3)$. □

However, since we call *RemoveDegree2Vertices* (Section 3.2), we will remove all the new hot pixels on s and the segments to its right, leaving the output in the above example with $\Theta(n^2)$ hot pixels. The reason is that the segments in the right structure are spread far enough such that it is impossible for one segment to snap to hot pixels lying on its neighboring segment. This example demonstrates that applying *RemoveDegree2Vertices* is very useful for improving the output quality.

4.2. Implementation issues

In this section we discuss the precision required to implement Snap Rounding and its variants (Section 4.2.1) and the geometric data structures we use (Trapezoidal decomposition and Dynamic simplex range searching in Sections 4.2.2 and 4.2.3 respectively).

4.2.1. Precision requirements of snap rounding

As far as we know, previous SR algorithms (except the one we mention below) required exact precision and no proof for ability of carrying out the work with finite-precision has been presented. For example, computing the exact location of an intersection of two segments (in order to locate the pixel to heat) will require exact arithmetic since it involves division.[6] Subsequently, ISR and ISRBD require exact arithmetic too as being extensions of SR. Most recently, Bhattacharya and Sember [3] claimed that their Snap Rounding algorithms can works with integer arithmetic. However, it is not clear how to use their ideas in other SR algorithms (including ISR and ISRBD) and whether it is possible. We believe that exploring the possibilities of implementing various SR algorithms with finite-precision arithmetic could be an interesting direction for further research.

4.2.2. Trapezoidal decomposition

Recall that we use the trapezoidal decomposition data structure [6] twice. The first is with the forbidden loci $F(S,\delta)$ (denoted by Γ_1) and the second is with the input segments S (denoted by Γ_2).

We build Γ_1, a trapezoidal decomposition of $F(S,\delta)$ (see Section 3.1 for details). Each trapezoid in Γ_1 may be within forbidden locus of one or more segments. For each trapezoid t, we denote by $t \to W$ the list of these segments. We need an efficient data structure that queries the corresponding segments for a given trapezoid. By using a trapezoidal decomposition, we can efficiently locate the trapezoid that contains a center of hot pixel. Once we retrieve the trapezoid t, we perform queries between the hot pixel and the segments using $t \to W$.

If we store $t \to W$ explicitly, we will need $O(n^3)$ space (since there are $O(n^2)$ trapezoids, each of which contains a list of $O(n)$ segments). To save space, we use the following technique. Each trapezoid holds only two pointers: one to another trapezoid (denoted by p_1) and the other to a segment (denoted by p_2). Then the segments for each trapezoid can be retrieved by tracing trapezoids with p_1 and collecting the segments pointed to by p_2. Let Γ_1^i be the temporary trapezoidal decomposition, after inserting the forbidden locus of the ith segment. After inserting forbidden locus $(i+1)$, we can partition the trapezoids into three groups. The first group, G_1, contains trapezoids which did not undergo any change during iteration $(i+1)$. The second group, G_2, contains new trapezoids whose W does not contain the forbidden locus of segment $(i+1)$. The final group, G_3, contains new trapezoids whose W contains the forbidden locus of segment $(i+1)$. Then the values of p_1 and p_2 of each trapezoid will be determined in the following way. The data of the trapezoids in G_1 will not change. p_1 of any new trapezoid in G_3 will point to a trapezoid in G_2 and their p_2 will point to segment $(i+1)$. The idea is that trapezoids of G_3 contain the same forbidden loci as G_2 plus the new one of segment $(i+1)$, so a trapezoid in G_3 will point to a trapezoid in G_2, whose W are equivalent except from the new forbidden locus. For each new trapezoid in G_2, there is a trapezoid of Γ_1^i such that both have the same W. We copy the data from the trapezoid in Γ_1^i to its corresponding trapezoid in G_2.[7] Since maintaining the pointers p_1 and p_2 can be done locally, it does not increase the asymptotic time complexity.

To conclude, for each hot pixel h, we locate the trapezoid that contains it and find the segments as described above. These will be the set of segments whose forbidden loci contain h. The working storage is linear in the number of trapezoids. Since we build a trapezoidal decomposition of arrangement of $2n$ input trapezoids ($F_l(s,\delta)$ and $F_r(s,\delta)$ for each segment s), this data structure requires $O(n^2)$ working storage, $O(n^2 \log n)$ time for preprocessing, and each query will take $O(\log n + k)$ time where k is the output size.

The second trapezoidal decomposition is Γ_2—trapezoidal decomposition of the input segments S. The purpose of this data structure is to provide an efficient query mechanism that locates the segment that is the closest to a hot pixel h inside $\Delta(s,h)$. Then, a pixel on this segment will be heated (see Section 3.1 for details). The working storage and running time are the same as required for Γ_1.

4.2.3. Dynamic simplex range searching

We use this data structure (denoted by Ψ) for two purposes. The first is to check whether a triangle is empty or not, to decide if a pixel should be heated (see Section 2 for details). The second use of Ψ is to query whether a segment intersects hot pixels (in *RemoveDegree2Vertices*; see Section 3.2). The technique for this query was introduced in [14]:

[6] An alternative way would be to perform division with finite-precision applying binary search; however, it would require a special software routine.

[7] We ignore degenerate cases for the sake of clarity. Degenerate cases have different behavior, but they can be maintained by the above data structure in the same way traditional methods that use trapezoidal decomposition do.

we query hot pixel centers with polygons defined as the Minkowski sum of the queried segment and a unit pixel centered at the origin. We need dynamicity, as we need to support insertions and deletions of hot pixels. Let I_1 and I_2 be the number of original and new hot pixels respectively. Let $I = I_1 + I_2$. For theoretical bounds we use the data structure by Agarwal and Matousek [1]. For any $\varepsilon > 0$, it requires $O(I^{1+\varepsilon})$ space, the preprocessing time is $O(I_1^{1+\varepsilon})$ and each update takes $O(I^{\varepsilon})$ time. A general query takes $O(\log I + k) = O(\log n + k)$ where k is the query result size. Since we need to query emptiness, the query time becomes $O(\log n)$.

Remark. In practice, this data structure is difficult to implement. Instead, we use a *kd-trees* [6]. Since kd-trees query axis aligned rectangles, our query will use the axis-aligned bounding box of the polygons being queried, and filter out points that are not contained inside the polygons. We found this alternative very efficient in practice. *kd-trees* were also used to simplify the implementation in [14].

4.3. Analysis

In this section we analyze the running time and working storage ISRBD requires. Let us review the notations that we use. n is the number of input segments. I_1 is the number of original hot pixels and I is the total number of hot pixels (original and new). K is the list of degree-2 vertices that do not correspond to endpoints with size κ.

We analyze the complexities for the new two procedures of ISRBD.

Theorem 4.11. *GenerateNewHotPixels takes* $O(n^3(\log n + I^{\varepsilon}))$ *time and* $O(n^2 + I^{1+\varepsilon})$ *working storage for any* $\varepsilon > 0$.

Proof. We itemize the work done in this procedure:

- Building Γ_1 and Γ_2 takes $O(n^2 \log n)$ time and the data structures require $O(n^2)$ working storage.
- Building Ψ takes $O(I_1^{1+\varepsilon})$ time and requires $O(I^{1+\varepsilon})$ working storage for any $\varepsilon > 0$.
- For each hot pixel we locate the trapezoid of Γ_1 that contains it. Since there are I hot pixels, the total time is $O(I \log n)$.
- Each original hot pixel h can be the source of heating $O(n)$ new hot pixels, and during this process the total number of times they are within forbidden loci is at most $O(n)$. The reason is that each segment can be detected here at most $O(1)$ times as our analysis in Section 4.1 indicates. Since there are $O(n^2)$ original hot pixels, in total there are $O(n^3)$ pixel-segment pairs for which we do the following work. (a) For hot pixel h and segment s we check if $\Xi(s, h)$ is true (time $O(\log n)$). (b) If there is a need to heat a pixel, we find the segment on which the pixel is heated (time $O(\log n)$). (c) Update Ψ and the hot pixels queue Q if a pixel is heated (time $O(I^{\varepsilon})$). Thus, the total time required for this item is $O(n^3(\log n + I^{\varepsilon}))$.

Together, *GenerateNewHotPixels* requires $O(n^3(\log n + I^{\varepsilon}))$ time and $O(n^2 + I^{1+\varepsilon})$ working storage for any $\varepsilon > 0$ (note that $I = O(n^3)$). □

Theorem 4.12. *RemoveDegree2Vertices takes* $O(\kappa^2 \log n + \kappa I^{\varepsilon})$ *time for any* $\varepsilon > 0$.

Proof. Each degree-2 hot pixel which does not correspond to endpoints can be processed either when first traversed in K or after another hot pixel, which contains it in its list W, was removed. Thus, each such hot pixel may be tested κ times, resulting in $O(\kappa^2)$ total tests. Each query takes $O(\log n)$ time. Each removal of a pixel from Ψ takes $O(I^{\varepsilon})$ for any $\varepsilon > 0$. There are at most κ such removals. Together, *RemoveDegree2Vertices* takes $O(\kappa(\kappa \log n + I^{\varepsilon}))$ time for any $\varepsilon > 0$. □

The next theorem summarizes the work of ISRBD.

Theorem 4.13. *Given an arrangement of n segments, Iterated Snap Rounding with Bounded Drift requires* $O(n^3(\log n + I^{\varepsilon}) + \kappa(\kappa \log n + I^{\varepsilon}) + L^{\frac{2}{3}} I^{\frac{2}{3}+\varepsilon} + L)$ *time, and* $O(n^2 + I^{1+\varepsilon} + L^{\frac{2}{3}} I^{\frac{2}{3}+\varepsilon})$ *working storage for any* $\varepsilon > 0$, *where L is the overall number of links in the chains produced by the algorithm, I is the total number of pixels produced by the algorithm and* κ *is the number of degree-2 hot pixels which does not correspond to segment endpoints.*

Proof. The complexities are obtained by summing up the time and working storage in Theorems 4.11 and 4.12 with the complexity of ISR [14]. □

4.4. Determining the maximum deviation parameter

The maximum drift δ is a user-specified parameter. However, it cannot be arbitrarily small; clearly it must be at least the maximum size a segment can be rounded each time ($\frac{\sqrt{2}}{2}$). However, we stated in Section 3.1 that $\delta > \frac{3\sqrt{2}}{2}$, so this is the constrain a user must follow when choosing δ. Thus, the minimum value for δ is about three times the maximum deviation of SR ($\frac{\sqrt{2}}{2}$).

Clearly, there is an advantage in using smaller δ values with which the geometric similarity is better. However, our experiments (Section 6) indicate that bigger values usually reduce the number of new pixels being heated. This is logical, since larger values of δ results in larger $\Delta(s, h)$, which have a higher probability of containing hot pixel centers that block other segments from snapping into hot pixels. Thus, we have a trade off when determining δ. An interesting direction for future work is to establish solid optimization criteria for choosing δ.

5. Desirable properties

Recall that $A(S)$ is the arrangement of a set of planar segments S, and we define $A^*(S)$ to be the output obtained by either rounding method, depending on the context. Recently, de Berg et al. [5] listed four desirable properties for the output SR produces. We next explore these properties, fit them to ISR and ISRBD, and formulate a new one.

Two of the properties listed in [5] hold for the output of ISR and ISRBD as well. We list them below.

Property 5.1. *Fixed-precision representation*: *All vertices of $A^*(S)$ are at centers of grid squares.*

Proof. It follows from the way the rounding stage performs (recall that the same rounding procedure is used for both ISR and ISRBD). Since its results are equivalent to a finite series of rounding steps of SR (in which this property clearly holds), all vertices of $A^*(S)$ are at centers of grid squares. □

Property 5.2. *Topological similarity*: *There is a continuous deformation of the segments in S to their snap-rounded counterparts such that no segment ever crosses completely over a vertex of the arrangement.*

Proof. Once the set of hot pixels is fixed (after *GenerateNewHotPixels*), the rounding stage will satisfy this property [12]. □

The other two properties are modified in the context of ISR and ISRBD. The first one holds for ISRBD only. It follows from Theorem 3.1 and the discussion in Section 3.1.

Property 5.3. *Geometric similarity*: *For each input segment $s \in S$, its output lies within the Minkowski sum of s and a disc with radius δ centered at the origin. δ is determined by the user and must be larger than $\frac{3\sqrt{2}}{2}$ units.*

Note that in ISR the distance may be as large as $\Theta(n^2)$. The second modified property holds with modifications for both ISRBD and ISR. It differs in each. The correctness follows from the discussion in Section 3.2. We list them next.

Property 5.4. *Non-redundancy* (*ISRBD version*): *Any degree-2 vertex corresponds to either an endpoint of an original segment or to a vertex whose removal violates either the property of the separation between a vertex and a non-incident edge or the geometric similarity property.*

Since the geometric similarity in the output of ISR may be poor, we modify the above property for its sake.

Property 5.5. *Non-redundancy* (*ISR version*): *Any degree-2 vertex corresponds to either an endpoint of an original segment or to a vertex whose removal violates the property of the separation between a vertex and a non-incident edge.*

In addition, we formulate another property that holds for both ISR and ISRBD (but not for SR). It was proved in [14] for ISR and holds for ISRBD since the same rounding stage is performed in both. This property is desirable for robust usage of the rounded arrangement. (See Section 1 for more details on this issue.)

Property 5.6. *Separation between a vertex and a non-incident edge*: *The distance between a vertex and a non-incident edge is at least half of a unit-pixel.*

This property, together with the fixed-precision representation property, establish good robustness criteria for the output in order to avoid errors due to floating-point limitations.

To conclude, we see that ISRBD is the first method to satisfy both the desirable properties listed in [5] and the one we formulated above.

6. Experimental results

We implemented the algorithm using the CGAL library [4]. We tested the algorithm with many examples and present two here. For each, we display the input, output and the new hot pixels. We also provide statistics that help with understanding the quality of the output. Our conclusions from the experiments follow (see Tables 1 and 2).

6.1. Randomized input

The idea behind this experiment is to have congested input with many intersections—thus creating many hot pixels. The segment coordinates are chosen randomly inside a fixed box. Our tests were controlled by the number of segments and δ. We performed several dozens tests for each such pair. Figs. 12(a) and (b) illustrate a small example with 20 random segments. In this example, two pixels were heated in *GenerateNewHotPixels*. Note that the bottom new hot pixel was later removed by *RemoveDegree2Vertices*. The data in Table 1 imply the following: (a) The number of new hot pixels (after removing degree-2 vertices) was relatively small and decreased as δ increased. (b) Most of the heated pixels (Usually more than 80%) were cooled in *RemoveDegree2Vertices*. (c) We had an increase of 29–42% in the average total time in comparison to ISR. The largest average number of new hot pixels was produced with 300 segments. This is because more segments produced denser original hot pixel sets which blocked segments from potential snapping into hot pixels, while less segments had a sparse set of original hot pixels, thus less new ones were introduced.

Table 1
Results of random segment tests. Each result is in the form $X(Y)/Z$ where X is the average number of new hot pixels in the output, Y is the number of new hot pixels before applying *RemoveDegree2Vertices* and Z is the average extra time ISRBD needed in comparison with ISR (in %)

δ	Number of segments			
	200	300	500	1000
10	0.2(1.7)/35	0.2(1.6)/32	0.2(1.8)/37	0.1(0.9)/36
6	0.3(2.7)/40	0.6(5.3)/35	0.4(3.4)/35	0.3(2.3)/31
4	0.3(2.5)/38	0.9(8.7)/34	0.6(5.2)/29	0.5(4.2)/36
3	0.8(6.4)/39	1.3(12.0)/42	0.8(7.8)/32	0.6(6.12)/35
2	1.0(9.1)/33	1.6(13.6)/37	1.1(8.2)/35	0.8(7.6)/39

Table 2
Results of polygon visibility graph tests. The format is equivalent to the one in Table 1

δ	Number of edges			
	50	100	200	500
10	0.5(4.0)/30	1.2(11.8)/35	2.3(24.8)/35	3.5(30.4)/38
6	0.6(6.6)/35	1.5(12.6)/37	2.6(22.4)/34	3.7(34.4)/33
4	1.0(10.4)/41	1.8(14.2)/40	2.0(15.8)/32	4.0(34.0)/33
3	0.8(6.8)/39	1.6(15.3)/40	1.8(15.9)/38	4.2(47.88)/41
2	0.5(4.3)/33	1.4(12.0)/38	1.8(17.64)/40	3.8(30.1)/39

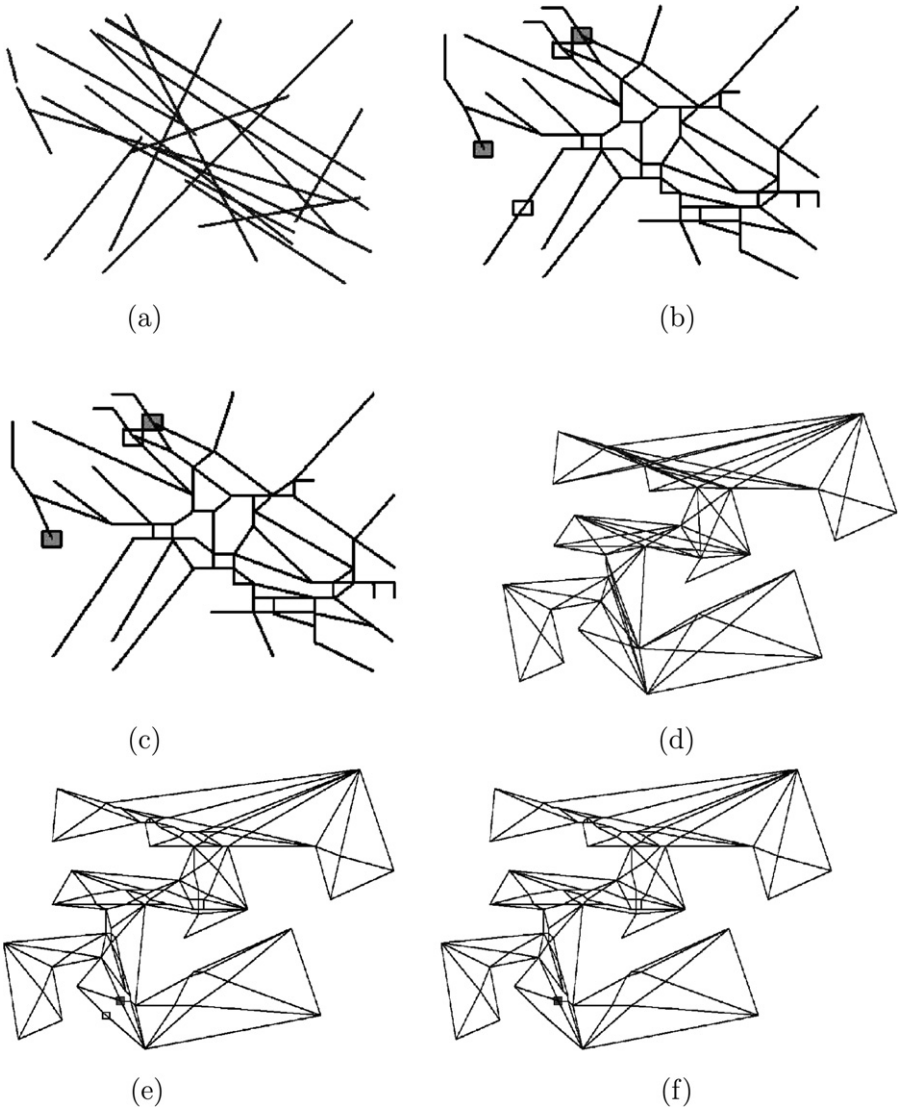

Fig. 12. Experiment snapshots obtained with our software: 20 random segments ((a) input, (b) output before removing redundant vertices, (c) final output) and a polygon with its visibility graph ((d) input, (e) output before removing redundant vertices, (f) final output). The new hot pixels are white and the hot pixels that are responsible for their heating are shaded.

6.2. Visibility graph of a polygon

We tested several examples of visibility graphs of polygons created randomly (we used the implementation of [2] for creating the polygons). We believe that such examples can be interesting since they may have many intersections in small area. Our testes were controlled by the number of edges in the polygons and δ. Table 2 summarizes the results, and Figs. 12(c) and (d) illustrate an example of a polygon with 31 segments. In this case, only one pixel was heated in *GenerateNewHotPixels* (it was later removed by *RemoveDegree2Vertices*). Our conclusions from this experiment is similar to the random segment experiment in both the number of new hot pixels and the extra time needed in ISRBD in comparison to ISR.

6.3. Experimental conclusions

We tested many examples, each with a variety of parameter values. Two experiment sets were presented here. In all examples, we observed similar behavior:

- The number of new hot pixels produced in *GenerateNewHotPixels* was very small in comparison to the number of original hot pixels. This is a positive indication for the output of ISRBD, since one generally prefers less hot pixels and fewer links in the output.
- The extra time required for ISRBD in comparison to ISR did not increase the order of the processing time.
- The majority of the pixels heated by *GenerateNewHotPixels* were removed in *RemoveDegree2Vertices* (see Section 3.2). Thus, it is useful to perform this optimization since it improves the output quality.

7. Conclusions

We presented an algorithm that rounds an arrangement of segments in 2D which follows Snap Rounding (SR) and Iterated Snap Rounding (ISR). It augments ISR with two simple and efficient procedures. The contribution of this algorithm is that it prevents problematic output that may be produced by SR or ISR. These occur in SR when there is a very small distance between vertices and non-incident edges, and in ISR when there is a large deviation of an approximating polygonal chain from the corresponding original segment. We eliminate these problematic features by introducing new hot pixels to ISR. We showed that ISRBD produces efficient output both in theory and in experimental evaluations. We also showed that ISRBD satisfies all the properties discussed in the literature so far, plus another one we formulated in this paper. Our experiments showed that relative to the number of original hot pixels, a very small number of new hot pixels is produced by our algorithm, and most of them are removed as they induced degree-2 vertices that meet certain conditions. The extra time required by our algorithm in comparison to ISR did not increase the order of the processing time. Based on these results, we believe that ISRBD is a good option for creating snap-rounded–like arrangements.

We propose a few directions for future research. First, we would like to further investigate the output complexity of our algorithm. We proved a tight output complexity, similar to SR and ISR, but there is still a gap when overlapping segments are counted only once. Since we experienced a small number of new hot pixels with our technique, it is also tempting to theoretically establish the upper bound and conditions for the amount of new hot pixels.

In Section 3.2 we showed how to remove redundant degree-2 vertices in order to simplify the output. It seems more efficient to avoid heating the corresponding pixels from the start. Thus, an interesting challenge would be to devise an efficient technique for identifying the redundancy of a pixel before heating it in *GenerateNewHotPixels*.

We explained the trade off in choosing δ (better geometric similarity vs. less new hot pixels). It would be interesting to establish good criteria for choosing δ, and possibly choose the value automatically based on the input.

Acknowledgements

The author would like to thank D. Halperin, J.S.B. Mitchell and A. Traeger for reviewing this work and providing helpful comments.

References

[1] K.P. Agarwal, J. Matousek, Dynamic half-space range reporting and its application, Technical report, Durham, NC, USA, 1991.
[2] T. Auer, M. Held, Heuristics for the generation of random polygons, in: Proc. 8th Canad. Conf. Comput. Geometry, 1996.
[3] B. Bhattacharya, J. Sember, Efficient snap rounding with integer arithmetic, in: Proc. 19th Canad. Conf. Comput. Geometry, 2007.
[4] The CGAL User Manual, Version 3.1, 2004; www.cgal.org.
[5] M. de Berg, D. Halperin, M. Overmars, An intersection-sensitive algorithm for snap rounding, Comput. Geom. Theory Appl. 36 (3) (2007) 159–165.
[6] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, Computational Geometry: Algorithms and Applications, Springer-Verlag, Heidelberg, 2004.
[7] O. Devillers, P. Guigue, Inner and outer rounding of set operations on lattice polygonal regions, in: Proc. 20th ACM Symposium on Computational Geometry, SoCG 2004, 2004, pp. 429–437.
[8] A. Eigenwillig, L. Kettner, N. Wolpert, Snap rounding of Bezier curves, in: Proc. 23rd ACM Symposium on Computational Geometry, SoCG, 2007.
[9] S. Fortune, Vertex-rounding a three-dimensional polyhedral subdivision, Discrete Comput. Geom. 22 (4) (1999) 593–618.
[10] M. Goodrich, L.J. Guibas, J. Hershberger, P. Tanenbaum, Snap rounding line segments efficiently in two and three dimensions, in: Proc. 13th Annu. ACM Sympos. Comput. Geom., 1997, pp. 284–293.
[11] D.H. Greene, F.F. Yao, Finite-resolution computational geometry, in: Proc. 27th Annu. IEEE Sympos. Found. Comput. Sci., 1986, pp. 143–152.

[12] L. Guibas, D. Marimont, Rounding arrangements dynamically, Internat. J. Comput. Geom. Appl. 8 (1998) 157–176.
[13] D. Halperin, E. Leiserowitz, Controlled perturbation for arrangements of circles, in: Proc. 19th ACM Symposium on Computational Geometry, SoCG, 2003, pp. 264–273.
[14] D. Halperin, E. Packer, Iterated snap rounding, Comput. Geom. Theory Appl. 23 (2) (2002) 209–225.
[15] D. Halperin, C.R. Shelton, A perturbation scheme for spherical arrangements with application to molecular modeling, Comput. Geom. Theory Appl. 10 (1998) 273–287.
[16] J. Hershberger, Improved output-sensitive snap rounding, in: Proc. 22nd ACM Symposium on Computational Geometry, SoCG, 2006, pp. 357–366.
[17] J. Hobby, Practical segment intersection with finite precision output, Comput. Geom. Theory Appl. 13 (1999) 199–214.
[18] V.J. Milenkovic, Shortest path geometric rounding, Algorithmica 27 (1) (2000) 57–86.
[19] E. Packer, Finite-precision approximation techniques for planar arrangements of line segments, Master's thesis, Dept. Comput. Sci., Tel-Aviv Univ., 2002.
[20] S. Raab, Controlled perturbation for arrangements of polyhedral surfaces with application to swept volumes, MSc. thesis, Dept. Comput. Sci., Bar Ilan University, Ramat Gan, Israel, 1999.
[21] C.K. Yap, Robust geometric computation, in: J.E. Goodman, J. O'Rourke (Eds.), Handbook of Discrete and Computational Geometry, Chapter 41.